```
/*  Project VendView
    SKY Wire, LP
    Copyright © 1994. All Rights Reserved.

    SUBSYSTEM:    vendview.exe Application
    FILE:       vndvstos.cpp
    AUTHOR:     Robert M. Cowling


    OVERVIEW
    ========
    Calculate Space TO Sales

*/
#pragma hdrfile "vndvwapp.csm"
#include "vndvwhdr.h"
#pragma hdrstop

typedef
  struct
    {
      int   code;
      int   removed;
      char  product[16];
      int   velocity;
      int   capacity;
      int   optimumCapacity;
      int   newCapacity;
    } BUTTONS;

typedef
  struct
    {
      int   capacity;
      int   assigned;
      int   newAssignment;
    } COLUMNS;


// MAX_BUTTONS is defined in vndvmdi5.h as 10
#define MAX_BUTTONS12 12
//#define MAX_COLS 20

#define MAX_S2SDAYS (-90) // -90 == 90 days back
#define MIN_S2SDAYS (-14)

// Paradox engine object and database object
//
extern BEngine    *dbEngine;
extern BDatabase  *dbDatabase;

// location of common databases
extern char szCommDir[];
extern char szMapDir[];

extern char szMachStatTableName[];// = "MACHSTAT"
extern char szFacilityTableName[];// = "FACILITY"
extern char szProductTableName[];// = "PRODUCT"
extern char szMachineLoadTable[];// = "MACHLOAD"

//
//
//
long  EvaluateFit(BUTTONS buttons[], int count)
  {
    int   index;
    long  answer = 0;

    for (index = 0; index < count; index++)
      {
        long  diff = buttons[index].optimumCapacity - buttons[index].newCapacity;
        long  prod = diff * diff;
        answer += prod;
      }
    return answer;
  }


#pragma argsused
//  Parameters passed:
//    Report code:  4 character report code - e.g. CASH - zero terminated string
//    Report path:  13 to 131 character path to report file - e.g.  VVRCASH.RPT
//    Print switch: True if print report
//    Display switch:  True if display report
//    Report title: 32 character report title - e.g. Cash Accountability
//    Report parms: from 1 to 16 parameters for report
//                  1.  Time of day to print report (HHMM)
//                  2.  Repeat code for day to print (bits = 000000000SSFTWTM)
//                  3.  From weeks (today +/- days)
//                  4.  To days (today +/- days) - not used
//                  5.  Amount  (two decimal positions implied)
//                  6.  Number of routes (0 = none, 99 = all)
//                  7.  First route
//                  8.  Second route
//                  9.  Third route, etc.
void GenerateS2SA(char *szReportPath, BOOL bPrint, BOOL bDisplay, char *szReportTitle, int
*nParms)
  {

     int   x, y, z;
     int   columns;
     int   buttons, oldbuttons;
     int   venderCapacity;
     long  venderVelocity;   // changed from int to long  RMC 2/6/96
     int   FromDays = -(nParms[2] * 7/* Days per week */ );
     int   minThreshold = nParms[4];
     int   maxProducts = nParms[6];
     int   nTmp;

     CHECKHANDLES();

     //
     // GET TODAY'S DATE, INCLUDING THE DAY OF THE WEEK.
     //

     // Get the dos date.
     struct dosdate_t today;
     _dos_getdate(&today);

     // Get it into a BDate also.
     BDate Today;
     BDate Search;
     Today.year = today.year;
     Today.month = today.month;
     Today.day = today.day;
     // make sure FromDays is negative, and MIN_S2SDAYS > FromDays > MAX_S2SDAYS
     FromDays = min(MIN_S2SDAYS, FromDays);
     FromDays = max(MAX_S2SDAYS, FromDays);
     // calculate search date
     IncrBDate(Today, FromDays, Search);
    // Get the dos date.
    struct dostime_t now;
    _dos_gettime(&now);
    if (now.hour > 7 && now.hour < 10)
      {
        // SGG Ask the user if they really want to run the report.
        int RunReport = BWCCMessageBox(GetFocus(),
          "Space to Sales analysis will invalidate any load sheets produced earlier today.
 Do you really want to do this now?" ,
          "VendView Space to Sales Analysis", MB_YESNO | MB_ICONQUESTION);
        if (RunReport != IDYES)
          return;

        // ask for the password
        int iRoute = 0;
        if(VendViewAskUserPasswordDlg(GetWindowPtr(GetActiveWindow()), &iRoute).Execute()
= IDOK)
          return;
      }

    // Make sure maxProducts is not ridiculously small, or greater than max number of butto
ns.
    if (maxProducts < 4)
      maxProducts = 4;
    if (maxProducts > MAX_BUTTONS12)
      maxProducts = MAX_BUTTONS12;


    struct
      {
        int nCode;
        char szName[17];
        int nAdds;          // additional products required
      } stProductName[100];
    int nProductCount;

    int typeorder[3][10]    = {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                               {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                               {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};
    int orderindex;
    int colagraphics;

    COLUMNS col[MAX_COLS];

    BUTTONS button[MAX_BUTTONS12];
    BUTTONS oldbutton[MAX_BUTTONS12];

    struct
      {
        int   capacity;
        int   column;
        int   assigned;
      } orderedCol[MAX_COLS + MAX_BUTTONS12 + MAX_BUTTONS12 + 1];

    struct
      {
        int   optimumCapacity;
        int   button;
        int   capacity;
      } orderedButton[MAX_BUTTONS12 + 1];

    char  szTable[MAXPATH];
    BOOL  bBlank; // blank field flag

    ////////////////////////////////////////////////////////////////
    //
    // READ PRODUCTS INTO ARRAY
    //
    //
    strcpy(szTable, szCommDir);
    strcat(szTable, szProductTableName);
    BCursor curProduct(dbDatabase, szTable);
    CHECKCURSOR(&curProduct);
    if ((curProduct.lastError == PXSUCCESS))
      {
        stProductName[0].nCode = 0;
        lstrcpy(stProductName[0].szName, "NONE ASSIGNED");
        nProductCount = 1;
        stProductName[0].nAdds = 0;
        curProduct.gotoBegin();
        do
          {
            curProduct.gotoNext();
            if (curProduct.lastError == PXSUCCESS)
              {
                curProduct.getRecord(); // retrieve found record
                ASSERT(curProduct.lastError == PXSUCCESS);
                BRecord *pRec = curProduct.genericRec;
                pRec->getField("Ident", stProductName[nProductCount].nCode, bBlank);
                ASSERT(pRec->lastError == PXSUCCESS);
                pRec->getField("Abbreviation", stProductName[nProductCount].szName, 17, bBl
ank);
                ASSERT(pRec->lastError == PXSUCCESS);
                if (bBlank)
                  stProductName[nProductCount].szName[0] = 0;
                // get ranking for brand selection on maximizing products

                pRec->getField("Flavor", orderindex, bBlank);
                ASSERT(pRec->lastError == PXSUCCESS);
                if (bBlank)
                  orderindex = 0;
                if ((orderindex > 0) && (orderindex < 11))
                  typeorder[0][orderindex - 1] = stProductName[nProductCount].nCode;
                pRec->getField("Standard", orderindex, bBlank);
                ASSERT(pRec->lastError == PXSUCCESS);
                if (bBlank)
                  orderindex = 0;
                if ((orderindex > 0) && (orderindex < 11))
                  typeorder[1][orderindex - 1] = stProductName[nProductCount].nCode;
                pRec->getField("Diet", orderindex, bBlank);
                ASSERT(pRec->lastError == PXSUCCESS);
                if (bBlank)
                  orderindex = 0;
                if ((orderindex > 0) && (orderindex < 11))
                  typeorder[2][orderindex - 1] = stProductName[nProductCount].nCode;

                stProductName[nProductCount].nAdds = 0;
                nProductCount++;
              }
          } while (curProduct.lastError == PXSUCCESS);
        curProduct.close();
      }
    //
    //
    // end of 'READ PRODUCTS INTO ARRAY'
    //
    ///////////////////////////////////////////////////////////////


    ///////////////////////////////////////////////////////////////
    //
    // CREATE CURSORS FOR PARADOX DATABASE ACCESS [
    //
    //

    // Build facility table cursor.
    strcpy(szTable, szCommDir);
    strcat(szTable, szFacilityTableName);
    BCursor curFacility(dbDatabase, szTable);
    CHECKCURSOR(&curFacility);

    // Build machine status table cursor.
    strcpy(szTable, szCommDir);
    strcat(szTable, szMachStatTableName);
    BCursor curMachStat(dbDatabase, szTable);
    CHECKCURSOR(&curMachStat);

    // Build vender load report auxiliary data table cursor.
    // This table contains strings needed for vender load report.
    // Most of the data in the vender load report is taken from MACHLOAD.DB.
    strcpy(szTable, szCommDir);
    strcat(szTable, "VND1LOAD.DB");
    BCursor curVenderLoad(dbDatabase, szTable);
    CHECKCURSOR(&curVenderLoad);

    // Build machine load cursor.
    strcpy(szTable, szCommDir);
    strcat(szTable, "MACHLOAD.DB");
    BCursor curMachLoad(dbDatabase, szTable);
    CHECKCURSOR(&curMachLoad);

    // Build space to sales load report auxiliary data table cursor.
    // This table contains strings needed for space to sales report.
    // EMPTY THE TABLE.
    EmptyTable("SP2SLOAD.DB");
    strcpy(szTable, szCommDir);
    strcat(szTable, "SP2SLOAD.DB");
    BCursor curSp2SLoad(dbDatabase, szTable);
    CHECKCURSOR(&curSp2SLoad);

    // Build space to sales total load cursor.
    // EMPTY THE TABLE.
    EmptyTable("SP2STOTL.DB");
    strcpy(szTable, szCommDir);
    strcat(szTable, "SP2STOTL.DB");
    BCursor curSp2STotal(dbDatabase, szTable);
    CHECKCURSOR(&curSp2STotal);

    //
    //
    // end of 'CREATE CURSORS FOR PARADOX DATABASE ACCESS' ]
    //
    ////////////////////////////////////////////////////////////////


    ////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////
    //
    //
    // generate space to sales print records
    //



    if (    curFacility.lastError == PXSUCCESS
         && curMachStat.lastError == PXSUCCESS
         && curVenderLoad.lastError == PXSUCCESS
         && curMachLoad.lastError == PXSUCCESS
```

```
        && curSp2SLoad.lastError == PXSUCCESS
        && curSp2STotal.lastError == PXSUCCESS)
      {
        FIELDNUMBER fld;
        BOOL  blank;
        BDate bdate;
        int   nRepnum;
        int   nextRoute;
        int   ident;
        int   facilIdent;
        int   routeIdent = 0;

        BRecord *machstatRec = curMachStat.genericRec;
        BRecord *facilRec = curFacility.genericRec;
        BRecord *vndloadRec = curVenderLoad.genericRec;
        BRecord *machloadRec = curMachLoad.genericRec;
        BRecord *s2sloadRec = curSp2SLoad.genericRec;
        BRecord *totalRec = curSp2STotal.genericRec;

        // fill column and button arrays

        curVenderLoad.gotoBegin();
        curVenderLoad.gotoNext();
        while (curVenderLoad.lastError == PXSUCCESS)
          {
            curVenderLoad.getRecord(vndloadRec);
            ASSERT(curVenderLoad.lastError == PXSUCCESS);

            // find machstat record
            vndloadRec->getField("Vender ident", ident, blank);
            ASSERT(vndloadRec->lastError == PXSUCCESS);
            if (blank)
              ident = 0;
            machstatRec->putField("Ident", ident);
            curMachStat.searchIndex(machstatRec, pxSearchFirst, 1);
            if (curMachStat.lastError == PXSUCCESS)
              {
                curMachStat.getRecord(machstatRec);

                // Skip if non-radio vender.
                machstatRec->getField("No radio", nTmp, blank);
                ASSERT(machstatRec->lastError == PXSUCCESS);
                if (blank)
                  nTmp = 0;
                if (nTmp)
                  {
                    curVenderLoad.gotoNext();
                    continue ;
                  }

                // Update Velocity for Machstat; skip vender if configuration of vender has
changed recently.
                if (!CalculateVelocityForVender(machstatRec, Today, Search))
                  {
                    curVenderLoad.gotoNext();
                    continue ;
                  }
                // find facility record
                machstatRec->getField("Facility Ident", facilIdent, blank);
                ASSERT(machstatRec->lastError == PXSUCCESS);
                if (blank)
                  facilIdent = 0;
                facilRec->putField("Ident", facilIdent);
                curFacility.searchIndex(facilRec, pxSearchFirst, 1);
                if (curFacility.lastError == PXSUCCESS)
                  {
```

```
curFacility.getRecord(facilRec);

// check for route change
facilRec->getField("Route Ident", nextRoute, blank);
if (blank)
  nextRoute = 0;
if ((routeIdent) && (routeIdent != nextRoute))  // change in routes
  {
    // write total load records
    // write out total additions for space to sales
    for (x = 0; x < nProductCount; x++)
      {
        if (stProductName[x].nAdds > 0)
          {
            totalRec->putField("Route", routeIdent);
            totalRec->putField("Product code", stProductName[x].nCode);
            totalRec->putField("Product name", stProductName[x].szName)
;
            totalRec->putField("Count", stProductName[x].nAdds);
            curSp2STotal.appendRec(totalRec);
            // reset count
            stProductName[x].nAdds = 0;
          }
      }

  }
routeIdent = nextRoute;

columns = 0;
buttons = 0;
venderCapacity = 0;
venderVelocity = 0;

// Init col and button arrays.
for (x = 0; x < MAX_COLS; x++)
  {
    col[x].capacity = 0;
    col[x].assigned = 0;
    col[x].newAssignment = 0;
  }
for (x = 0; x < MAX_BUTTONS12; x++)
  {
    button[x].code = 0;
    button[x].removed = 0;
    button[x].product[0] = 0;
    button[x].capacity  = 0;
    button[x].velocity  = 0;
    button[x].optimumCapacity = 0;
    button[x].newCapacity = 0;
  }

// Get product codes.
fld = machstatRec->getFieldNumber("Product 1 code");
if (machstatRec->lastError == 0)
  {
    for (x = 0; x < MAX_BUTTONS12; x++)
      {
        machstatRec->getField(fld + x, button[x].code, blank);

        if ((machstatRec->lastError) || (blank))
          button[x].code = 0;
        if (button[x].code)
          buttons = x + 1;
      }
  }

```

```
            // Get product names.
            for (x = 0; x < buttons; x++)
              {
                for (y = 0; y < nProductCount; y++)
                  {
                    if (button[x].code == stProductName[y].nCode)
                      break;
                  }
                if (y >= nProductCount)
                  y = 0;  // reset to no name
                strcpy(button[x].product, stProductName[y].szName);
              }

            // Get column capacities.
            fld = machstatRec->getFieldNumber("Column 1 capacity");
            if (machstatRec->lastError == 0)
              {
                for (x = 0; x < MAX_COLS; x++)
                  {
                    machstatRec->getField(fld + x, col[x].capacity, blank);

                    if ((machstatRec->lastError) || (blank))
                      col[x].capacity = 0;
                    if (col[x].capacity)
                      {
                        columns = x + 1;
                        venderCapacity += col[x].capacity;
                        // fill in ordered array
                        orderedCol[x].capacity = col[x].capacity;
                        orderedCol[x].column = x + 1;  // column no.
                        orderedCol[x].assigned = 0;
                      }
                  }
              }

            // Get column assignments and button capacities,
            fld = machstatRec->getFieldNumber("Column 1 assigned");
            if (machstatRec->lastError == 0)
              {
                for (x = 0; x < columns; x++)
                  {
                    machstatRec->getField(fld + x, col[x].assigned, blank);

                    if ((machstatRec->lastError) || (blank))
                      col[x].assigned = 0;

                    col[x].newAssignment = col[x].assigned;

                    // if assignment is valid, add to button capacity
                    if ((col[x].assigned > 0) && (col[x].assigned <= buttons))
                    {
                      button[col[x].assigned - 1].capacity += col[x].capacity;
                      button[col[x].assigned - 1].newCapacity += col[x].capacity;
                      // fill in ordered col array
                      orderedCol[x].assigned = col[x].assigned;
                    }

                  }
              }
            fld = machstatRec->getFieldNumber("Product 1 velocity");
            if (machstatRec->lastError == 0)
              {
                for (x = 0; x < buttons; x++)
                  {
                    machstatRec->getField(fld + x, button[x].velocity, blank);

            if ((machstatRec->lastError) || (blank))
              button[x].velocity = 0;
            venderVelocity += button[x].velocity;
            // adjust dual assignments
            if ((x) && (button[x].code) &&
                (button[x].code == button[x - 1].code) &&
                (button[x - 1].capacity == 0))
              {
                button[x].velocity += button[x - 1].velocity;
                button[x - 1].velocity = 0;
              }
          }
      }

    // Save a copy of the original button configuration.

    for (x = 0; x < buttons; x++)
      {
        oldbutton[x].code = button[x].code;
        strcpy(oldbutton[x].product, button[x].product);
        oldbutton[x].capacity  = button[x].capacity;
        oldbutton[x].velocity  = button[x].velocity;
      }
    oldbuttons = buttons;

    // Check for candidate vender where number of products is greater than
min products.
    // Get range of sales for setting maxProducts (number of recommended br
ands) RMC 2/6/96
    int  products = 0;

    for (x = 0; x < buttons; x++)
      if (button[x].capacity > 0)
        products++;

    // velocity total is per day (365 * 6.58 = ~100 cases per year)
      maxProducts = 5;  // minimum maximum -- < 100    = 5
    if (venderVelocity > 658)           // 100 - 200 = 6
      maxProducts++;
    if (venderVelocity > 1315)          // 200 - 300 = 7
      maxProducts++;
    if (venderVelocity > 1973)          // > 300     = 8
      maxProducts++;

    while (products > (maxProducts - 1))
      {
        // check for product velocity below threshold
        // get minimum product with minimum velocity
        int   minVelocity = 9999;
        int   productWithMin;
        for (x = 0; x < buttons; x++)
          {
            if (button[x].capacity > 0 && button[x].velocity <= minVelocity
)
              {
                minVelocity = button[x].velocity;
                productWithMin = x;
              }
          }

        if (minVelocity < minThreshold)
          {
            venderVelocity -= button[productWithMin].velocity;

            // Get count of buttons in group to be removed.
            for (int nCountOfGroup=1; nCountOfGroup<=productWithMin; nCount
CfGroup++)
                              if (     button[productWithMin-nCountOfGroup].code != button[productWithMin].code
                                  ||  button[productWithMin-nCountOfGroup].capacity > 0)
                                break;

                            // Give the group's columns to button 1. All columns should be assigned to
                            // the last button in the group.
                            button[0].newCapacity += button[productWithMin].newCapacity;
                            for (x=0; x<columns; x++)
                              if (col[x].assigned == productWithMin + 1)
                                orderedCol[x].assigned = col[x].newAssignment = col[x].assigned = 1;

                            // Shift the remaining buttons below the group up.
                            for (x = productWithMin + 1; x < buttons; x++)
                              {
                                button[x-nCountOfGroup].code = button[x].code;
                                strcpy(button[x-nCountOfGroup].product, button[x].product);
                                button[x-nCountOfGroup].capacity = button[x].capacity;
                                button[x-nCountOfGroup].velocity = button[x].velocity;
                                button[x-nCountOfGroup].optimumCapacity = button[x].optimumCapacity;
                                button[x-nCountOfGroup].newCapacity = button[x].newCapacity;
                                for (y=0; y<columns; y++)
                                  if (col[y].assigned == x + 1)
                                    orderedCol[y].assigned = col[y].newAssignment = col[y].assigned = x-nCountOfGroup+1;
                              }

                            // Empty the last nCountOfGroup buttons.
                            for (x=buttons-nCountOfGroup;x<buttons;x++)
                              {
                                button[x].code = 0;
                                button[x].product[0] = 0;
                                button[x].capacity = 0;
                                button[x].velocity = 0;
                                button[x].optimumCapacity = 0;
                                button[x].newCapacity = 0;
                              }

                            // Reduce the number of buttons by nCountOfGroup.
                            buttons -= nCountOfGroup;
                            products--;
                          }
                        else
                          break;
                      }

                    int baseset = (maxProducts + 3) / 2;  // 4 or 5
                    // recommended product list  (from typeorder  -code:flag)
                    int productlist[10][2];

                    // check for too few products and add from standard list
                    if (products < maxProducts)  // need to add products
                      {
                        // what kind is this vender? flavor, standard, or diet
                        // get top 4 selling products from this vender
                        // save brand and velocity for top selling four
                        int  topseller[4][2] = {{0, 0}, {0, 0}, {0, 0}, {0, 0}};
                        for (x = 0; x < products; x++)
                          {
                            if (button[x].velocity > topseller[0][1])
                              {
```

```
        topseller[3][0] = topseller[2][0];
        topseller[3][1] = topseller[2][1];
        topseller[2][0] = topseller[1][0];
        topseller[2][1] = topseller[1][1];
        topseller[1][0] = topseller[0][0];
        topseller[1][1] = topseller[0][1];
        topseller[0][0] = button[x].code;
        topseller[0][1] = button[x].velocity;
      }
    else if (button[x].velocity > topseller[1][1])
      {
        topseller[3][0] = topseller[2][0];
        topseller[3][1] = topseller[2][1];
        topseller[2][0] = topseller[1][0];
        topseller[2][1] = topseller[1][1];
        topseller[1][0] = button[x].code;
        topseller[1][1] = button[x].velocity;
      }
    else if (button[x].velocity > topseller[2][1])
      {
        topseller[3][0] = topseller[2][0];
        topseller[3][1] = topseller[2][1];
        topseller[2][0] = button[x].code;
        topseller[2][1] = button[x].velocity;
      }
    else if (button[x].velocity > topseller[3][1])
      {
        topseller[3][0] = button[x].code;
        topseller[3][1] = button[x].velocity;
      }
  }
// which type matches closest  using typeorder
int typescore[3] = {0, 0, 0};
// score each type
for (x = 0; x < 3; x++)
  {
    // score each top seller within type
    for (y = 0; y < 4; y++)
      {
        // score for each brand / rank for top seller within type
        for (z = 0; z < 4; z++)
          {
            if (topseller[y][0] == typeorder[x][z])
              typescore[x] += (4 - y) * (4 - z);
          }
      }
  }
  // pick winner
colagraphics = 0;
if (typescore[1] > 0)
  {
    colagraphics = 1;  // preset to standard
    if ((typescore[0] > typescore[1]) && (typescore[0] > typescore
2]))
      colagraphics = 0;
    if ((typescore[2] > typescore[1]) && (typescore[2] > typescore
0]))
      colagraphics = 2;

    // set suggested product list; flag unusable codes
    for (x = 0; x < 10; x++)
      {
        productlist[x][0] = typeorder[colagraphics][x];
        // enter initial velocity
        productlist[x][1] = venderVelocity / (x + 5);

```

```
            // check if used
            for (y = 0; y < oldbuttons; y++)
              {
                if (oldbutton[y].code == productlist[x][0])
                  {
                    // mark out all previously used products
                    // check if just removed
                    for (z = 0; z < buttons; z++)
                      {
                        if (button[z].code == productlist[x][0])
                          break;
                      }
                    if (z == buttons)
                      productlist[x][1] = -2;  // used but dropped flag
                    else
                      productlist[x][1] = -1;  // used flag
                    break;
                  }
              }
          }
      }
  }

// if too few products, add preferred brands
if (colagraphics)
  {
    while ((products < maxProducts) && (buttons < MAX_BUTTONS12))
      {
        int newproduct[2] = {0, 0}; // code;velocity
        int newproductindex = 0;

        // find suggested brand
        for (x = 0; x < maxProducts; x++)
          {
            // scan suggested list for available products
            if (productlist[x][1] > 0)
              {
                newproduct[0] = productlist[x][0];
                newproduct[1] = productlist[x][1];
                newproductindex = x;

                // break out if basic product
                if (x < baseset)
                  break;
              }
            else  // flag is -1 (used) or -2 (dropped)
              {
                // if there is a suggested product and it replaces
                // a dropped product, use it
                if ((newproduct[0]) && (productlist[x][1] == -2))
                  break;
              }
          }
        // exit sentinel -- break if no new products
        if (newproduct[0] == 0)
          break;
        else
          {
            productlist[newproductindex][1] = -3;  // using suggestion
          }

        // fill in new product
        button[buttons].code = newproduct[0];
        // find product name

        for (x = 0; x < nProductCount; x++)
```

```
          {
            if (stProductName[x].nCode == newproduct[0])
              break;
          }
        if (x >= nProductCount)
          x = 0;
        strcpy(button[buttons].product, stProductName[x].szName);
        button[buttons].capacity = 1;    // mark as not dually assigned
        button[buttons].velocity = newproduct[1];
        venderVelocity += newproduct[1];
        button[buttons].optimumCapacity = 0;
        button[buttons++].newCapacity = 0;
        products++;
      }
  }

// calculate optimum capacity
for (x = 0; x < buttons; x++)
  {
    long opt = 0L;
    if (button[x].capacity > 0)  // not dual assigned
      {
        opt = button[x].velocity;
        opt *= venderCapacity;
        if (venderVelocity > 0)
          opt /= venderVelocity;
        else
          opt = 1;
        if (opt <= 0)
           opt = 1L;
      }
    button[x].optimumCapacity = (int) opt;
    orderedButton[x].optimumCapacity = (int) opt;
    orderedButton[x].button = x + 1;  // button no.
    orderedButton[x].capacity = 0;
  }
// order button array
for (x = buttons - 1; x > 0; x--)
  {
    for (y = 0; y < x; y++)
      {
        if (orderedButton[y].optimumCapacity > orderedButton[y + 1].opt
imumCapacity)
          {
            //  swap high for low
            orderedButton[MAX_BUTTONS12] = orderedButton[y];
            orderedButton[y] = orderedButton[y + 1];
            orderedButton[y + 1] = orderedButton[MAX_BUTTONS12];
          }
      }
  }

// add two dummy columns with zero capacity for each button
for (x = 0; x < buttons; x++)
  {
    orderedCol[columns].capacity = 0;
    orderedCol[columns].column = 0;
    orderedCol[columns++].assigned = x + 1;
    orderedCol[columns].capacity = 0;
    orderedCol[columns].column = 0;
    orderedCol[columns++].assigned = x + 1;
  }

// order column array
for (x = columns - 1; x > 0; x--)
  {
```

```
                        for (y = 0; y < x; y++)
                          {
                            if (orderedCol[y].capacity > orderedCol[y + 1].capacity)
                              {
                                // swap high for low
                                orderedCol[MAX_COLS + MAX_BUTTONS12 + MAX_BUTTONS12] = orde
redCol[y];
                                orderedCol[y] = orderedCol[y + 1];
                                orderedCol[y + 1] = orderedCol[MAX_COLS + MAX_BUTTONS12 + M
AX_BUTTONS12];
                              }
                          }
                      }

                    //get initial fit with old assignments
                    long fit = EvaluateFit(button, buttons);
                    long savefit = fit;

                    // initial new assignments
                    y = buttons - 1;
                    for (x = columns - 1; x >= 0; x--)
                      {
                        // if button is Dually Assigned with following button(s), skip it.
                        if (button[y].optimumCapacity == 0)
                          x++;
                        else
                          {
                            // unassign original, reassign
                            int asn = orderedCol[x].assigned;
                            orderedCol[x].assigned = y + 1;  // new button assignment
                            col[orderedCol[x].column - 1].newAssignment = y + 1;
                            // adjust capacities
                            button[orderedCol[x].assigned - 1].newCapacity += orderedCol[x]
.capacity;
                            button[asn - 1].newCapacity -= orderedCol[x].capacity;
                          }
                        // Move back one button; wrap when needed.
                        y--;
                        if (y < 0)
                          y = buttons - 1;
                      }

                    // swap routine
                    //    swap each column with all others from smallest to largest
                    savefit = EvaluateFit(button, buttons);
                    for (z = 0; z < 10; z++)
                      {
                        long keepfit = savefit;  // check in loop if optimization has been
reached
                        for (x = columns - 1; x > 0; x--)
                          {
                            for (y = x - 1; y >= 0; y--)
                              {
                                // swap capacities if different
                                if (orderedCol[x].capacity != orderedCol[y].capacity)
                                  {
                                    // if neither button is dually assigned
                                    if (    button[orderedCol[x].assigned - 1].optimumCapac
ity > 0
                                        &&  button[orderedCol[y].assigned - 1].optimumCapac
ity > 0)
                                      {
                                        // okay they are different, now swap and evaluate
                                        button[orderedCol[x].assigned - 1].newCapacity -= o
rderedCol[x].capacity;
                                        button[orderedCol[x].assigned - 1].newCapacity += o
rderedCcl(y).capacity;
                          button[orderedCol[y].assigned - 1].newCapacity += o
rderedCol[x].capacity;
                          button[orderedCol[y].assigned - 1].newCapacity -= o
rderedCol[y].capacity;
                          long tryfit = EvaluateFit(button, buttons);

                          // make sure buttons have some capacity after trade
                          if (      tryfit < savefit
                              &&  button[orderedCol[x].assigned - 1].newCap
acity > 0
                              &&  button[orderedCol[y].assigned - 1].newCap
acity > 0)
                            {
                             savefit = tryfit;

                             // adjust records
                             int asn = orderedCol[x].assigned;
                             orderedCol[x].assigned = orderedCol[y].assigned
;
                             orderedCol[y].assigned = asn;
                             col[orderedCol[x].column - 1].newAssignment = o
rderedCol[x].assigned;
                             col[orderedCol[y].column - 1].newAssignment = o
rderedCol[y].assigned;
                            }
                           else  // set capacities back to original setting
                            {
                             button[orderedCol[x].assigned - 1].newCapacity
+= orderedCol[x].capacity;
                             button[orderedCol[x].assigned - 1].newCapacity
-= orderedCol[y].capacity;
                             button[orderedCol[y].assigned - 1].newCapacity
-= orderedCol[x].capacity;
                             button[orderedCol[y].assigned - 1].newCapacity
+= orderedCol[y].capacity;
                            }
                         }
                      }
                    }
                 }
                if (savefit == keepfit) // no change in value
                  break;
               }
             columns -= (2 * buttons);  // subtract out dummy columns count

             // attempt to keep original column assignments if equal capacity
             for (x = 0; x < columns; x++)
               // if column x has moved to a different button
               if (col[x].assigned != col[x].newAssignment)
                 // Look for a column y with same capacity as column x
                 // now assigned to column x's original button,
                 // and not originally assigned to that same button.
                 for (y = 0; y < columns; y++)
                   if (     x != y
                        &&  col[x].capacity == col[y].capacity
                        &&  col[x].assigned == col[y].newAssignment
                        &&  col[y].assigned != col[y].newAssignment)
                     {
                      // Swap columns x and y.
                      col[y].newAssignment = col[x].newAssignment;
                      col[x].newAssignment = col[x].assigned;
                     }

             // calculate effectiveness
             long  origService = 10000L;
```

```
            long  newService  = 10000L;
            long  workService;

            for (x = 0; x < buttons; x++)
              {
                if ((button[x].velocity > 20) && (button[x].capacity > 1))
                  {
                    workService = button[x].capacity;
                    workService *= 10000;
                    workService /= button[x].velocity;
                    if (workService < origService)
                      origService = workService;

                    workService = button[x].newCapacity;
                    workService *= 10000;
                    workService /= button[x].velocity;
                    if (workService < newService)
                      newService = workService;
                  }
              }

            // Put removed products back in.
            for (x = 0; x < oldbuttons; x++)
              if (button[x].code != oldbutton[x].code)
                {
                  for (y = buttons; y > x; y--)
                    {
                      button[y].code = button[y-1].code;
                      strcpy(button[y].product, button[y-1].product);
                      button[y].velocity = button[y-1].velocity;
                      button[y].capacity = button[y-1].capacity;
                      button[y].optimumCapacity = button[y-1].optimumCapacity;
                      button[y].newCapacity = button[y-1].newCapacity;
                    }
                  buttons++;
                  button[x].code = oldbutton[x].code;
                  button[x].removed = -1;
                  strcpy(button[x].product, oldbutton[x].product);
                  button[x].velocity = oldbutton[x].velocity;
                  button[x].capacity = oldbutton[x].capacity;
                  button[x].optimumCapacity = 0;
                  button[x].newCapacity = 0;
                }

            /*
           ASSERT(buttons == oldbuttons);
           for (x = 0; x < oldbuttons; x++)
             {
               ASSERT(button[x].code == oldbutton[x].code);
               ASSERT(button[x].velocity == oldbutton[x].velocity);
               ASSERT(button[x].capacity == oldbutton[x].capacity);
             }
            */
            // write out results
            if (((origService < 400) && (newService > (origService + 100))) ||
                ((origService < 700) && (newService > (origService + 300))) ||
                (newService > (origService + 800)))
              {
                vndloadRec->getField("Report number", nRepnum, blank);
                if (blank)
                  nRepnum = 0;

                machloadRec->putField("Report number", nRepnum);
                curMachLoad.searchIndex(machloadRec, pxSearchFirst, 1);
                if (curMachLoad.lastError == PXSUCCESS)
                  {
```

```
            curMachLoad.getRecord(machloadRec);
            ASSERT(curMachLoad.lastError == PXSUCCESS);

            char  szWork[33];

            FIELDNUMBER fillEstField = machloadRec->getFieldNumber("Product 1 estimated");
            FIELDNUMBER fillS2sField = machloadRec->getFieldNumber("Product 1 s2s");
            FIELDNUMBER codeS2sField = machloadRec->getFieldNumber("Product 1 code s2s");
            FIELDNUMBER buttonS2sField = machloadRec->getFieldNumber("Button 1 assigned s2s");
            FIELDNUMBER columnS2sField = machloadRec->getFieldNumber("Column 1 assigned s2s");

            FIELDNUMBER buttonListField  = s2sloadRec->getFieldNumber("Product 1 button list");
            FIELDNUMBER columnListField  = s2sloadRec->getFieldNumber("Product 1 column list");
            FIELDNUMBER velocityField    = s2sloadRec->getFieldNumber("Product 1 velocity");
            FIELDNUMBER capacityEstField = s2sloadRec->getFieldNumber("Product 1 capacity est");
            FIELDNUMBER capacityOptField = s2sloadRec->getFieldNumber("Product 1 capacity opt");
            FIELDNUMBER capacitys2sField = s2sloadRec->getFieldNumber("Product 1 capacity s2s");
            FIELDNUMBER productnameField = s2sloadRec->getFieldNumber("Product 1 name");

            s2sloadRec->putField("Vender ident", ident);
            s2sloadRec->putField("Delivery date", bdate);
            s2sloadRec->putField("Report number", nRepnum);

            // Init all array fields in machload record to blank.
            for (x = 0; x < MAX_BUTTONS; x++)
              {
                machloadRec->setNull(fillS2sField + x);
                machloadRec->setNull(codeS2sField + x);
                machloadRec->setNull(buttonS2sField + x);
              }
            for (x = 0; x < MAX_COLS; x++)
              machloadRec->setNull(columnS2sField + x);

            // Init all array fields in sp2sload record to blank.
            for (x = 0; x < MAX_BUTTONS; x++)
              {
                s2sloadRec->setNull(buttonListField + x);
                s2sloadRec->setNull(productnameField + x);
                s2sloadRec->setNull(columnListField + x);
                s2sloadRec->setNull(velocityField + x);
                s2sloadRec->setNull(capacityEstField + x);
                s2sloadRec->setNull(capacityOptField + x);
                s2sloadRec->setNull(capacitys2sField + x);
              }


            // Write s2s column assignments to machload record.
            for (x = 0; x < MAX_COLS; x++)
              machloadRec->putField(columnS2sField + x, col[x].newAssignment);

            // Calc delta capacities, get est fill amounts, and calc s2s fill amounts.
            int nDiff[MAX_BUTTONS12];
```

```
int nUnroundedS2sFill[MAX_BUTTONS12];
int nProductCounter = 0;
for (x = 0; x < buttons; x++)
  {

    // only for last button of each product
    if (button[x].capacity > 0)
      {
        // Calculate delta capacity and save in delta fill array nDiff.
        if (button[x].capacity == 1)
          nDiff[nProductCounter] = button[x].newCapacity;
        else
          nDiff[nProductCounter] = button[x].newCapacity - button[x].capacity;

        // S2s fill = est fill + delta capacity.
        machloadRec->getField(fillEstField + nProductCounter, nUnroundedS2sFill[nProductCounter], blank);
        if (blank)
          nUnroundedS2sFill[nProductCounter] = 0;
        nUnroundedS2sFill[nProductCounter] += nDiff[nProductCounter];
        nProductCounter++;
      }
  }

// Round non-negative fill amounts down to nearest whole 6-pack.
// Note that x here iterates over products, not buttons!
int nS2sFill[MAX_BUTTONS12];
int nTotalNonNegFill = 0;
for (x = 0; x < nProductCounter; x++)
  if (nUnroundedS2sFill[x] > 0)
    {
      // Round down to nearest multiple of 6. Result will be >= 0.
      nS2sFill[x] = nUnroundedS2sFill[x] / 6 * 6;
      // Add rounded-off fill to total of non-negative fills.
      nTotalNonNegFill += nS2sFill[x];
    }
  else
    nS2sFill[x] = nUnroundedS2sFill[x];

// Decrement 2s2 non-neg fills by 6 until we have even cases on trip from truck to vender.
// Note that x here iterates over products, not buttons!
//
// Dependencies on variables set outside this while loop:
//  (d1) Any positive values in the first nProductCounter elements of array nS2sFill must be multiples of 6.
//  (d2) nTotalNonNegFill must contain the sum of those positive values referred to in (1).
// Proof that while loop will halt:
//  (1) By observation of while loop conditional expression, the while loop halts when
//      nTotalNonNegFill is a multiple of 24 between iterations.
//  (2) We know that zero is a multiple of 6 and of 24.
//  (3) By (d2), nTotalNonNegFill is the sum of all nS2sFill > zero.
//  (4) By (3), nTotalNonNegFill is always >= zero.
//  (5) By (1) and (2), when nTotalNonNegFill is zero between iterations, while loop will halt.
//  (6) By (3), when nTotalNonNegFill > zero, there is at least one nS2sFill > 0.
// (7) By (4), (5) and (6), between iterations, either the whole loop will halt or it will enter the
//      next iteration guaranteed to have at least one nS2sFill > 0.
// (8) By (d1), all nS2sFill > zero were rounded down to nearest multiples of 6.
// (9) We know that the sum of multiples of an integer is a multiple of that integer.
// (a) By (3), (8) and (9), nTotalNonNegFill is always a multiple of 6.
// (b) By (6) and observation of code in body of while loop, nTotalNonNegFill must be decremented
//      at least once in each iteration of the while loop.
// (c) We know that every 4th multiple of 6 is a multiple of 24, so it takes at most 3 iterations
//      to go from any multiple of 6 to a multiple of 24 by subtracting 6 each iteration.
// (d) By observation of conditional expression in for loop header,
//      if nTotalNonNegFill reaches a multiple of 24 before the last iteration of the
//      for loop, the for loop will exit without further decrementing nTotalNonNegFill,
//      allowing the while loop conditional expression to halt the while loop.
// (e) By (b), (c) and (d), the while loop must halt in at most 3 iterations.
//
while (nTotalNonNegFill % 24)
  for (x = 0; x < nProductCounter && nTotalNonNegFill % 24; x++)
    if (nS2sFill[x] > 0)
      {
        // Decrement both nS2sFill and nTotalNonNegFill.
        nS2sFill[x] -= 6;
        nTotalNonNegFill -= 6;
      }

// Decrease nDiffs by decreases in fills due to rounding.
// Note that x here iterates over products, not buttons!
for (x = 0; x < nProductCounter; x++)
  nDiff[x] -= nUnroundedS2sFill[x] - nS2sFill[x];

// Write various data for buttons and products.
nProductCounter = 0;
int nVacatedButtonCounter = 0; // counter for shifting product up into vacated buttons
int oldMinDays = 32767;
int newMinDays = 32767;
for (x = 0; x < buttons; x++)
  {

    // if button not being vacated
    if (!button[x].removed)
      // Write the 'product number' to which this button is 'assigned'.
      machloadRec->putField(buttonS2sField+x-nVacatedButtonCounter, nProductCounter+1);

    // Write product values out only for last button of each product.
    if (button[x].capacity > 0)
      {
        // Save positive additional capacities in product array.
        if (nDiff[nProductCounter] > 0)  // only for positive a
```

```
dditional fill
  for (y = 0; y < nProductCount; y++)
    if (button[x].code == stProductName[y].nCode)
      {
        stProductName[y].nAdds += nDiff[nProductCounter];
        break;
      }

// Write s2s fill amount to machload rec.
machloadRec->putField(fillS2sField + nProductCounter, nS2sFill[nProductCounter]);

// Write product code to machload rec.
machloadRec->putField(codeS2sField + nProductCounter, button[x].code);


// Get old days and new days, and update oldMinDays and newMinDays
int days;

if (button[x].capacity > 1)
  {
    days = MulDiv(button[x].capacity, 10000, button[x].velocity);
    if (days < oldMinDays)
      oldMinDays = days;
  }
// Only update newMinDays if product not being removed.
if (!button[x].removed)
  {
    days = MulDiv(button[x].newCapacity, 10000, button[x].velocity);
    if (days < newMinDays)
      newMinDays = days;
  }

// Write button string  (eg. "2, 3, 4") to sp2sload rec.
// For removed low-vel products, write "Remove".
// Shift button numbers for remaining products up (lower nos.)
// to what they will be after removal of low-vel products.
if (button[x].removed)
  strcpy(szWork, "Remove");
else
  for (y = 0; y <= x; y++)
    {
      char  szBut[5];
      wsprintf(szBut, ", %d", y + 1 - nVacatedButtonCounter);
      if ((y) &&
          (button[y].code == button[y - 1].code) &&
          (button[y - 1].capacity == 0))
        strcat(szWork, szBut);
      else
        strcpy(szWork, &szBut[2]);
    }
s2sloadRec->putField(buttonListField + nProductCounter, szWork); // (new buttons)

// Write vel, cap, opt cap and new cap to sp2sload rec.
// Opt cap and new cap will be zero for removed products.
s2sloadRec->putField(velocityField + nProductCounter, b
utton[x].velocity);
                   int buttoncapacity = (button[x].capacity == 1) ? 0 : button[x].capacity;
                   s2sloadRec->putField(capacityEstField + nProductCounter, buttoncapacity);
                   s2sloadRec->putField(capacityOptField + nProductCounter, button[x].optimumCapacity);
                   s2sloadRec->putField(capacitys2sField + nProductCounter, button[x].newCapacity);
                   s2sloadRec->putField(productnameField + nProductCounter, button[x].product);

                   // write product's column list to sp2sload rec.
                   if (button[x].removed)
                     s2sloadRec->setNull(columnListField + nProductCounter);
                   else
                     {
                       szWork[0] = 0;
                       if (button[x].newCapacity)
                         for (y = 0; y < columns; y++)
                           {
                             if (col[y].newAssignment == x + 1 - nVacatedButtonCounter)
                               {
                                 char  szCol[5];

                                 wsprintf(szCol, ", %d", y + 1);
                                 if (szWork[0])
                                   strcat(szWork, szCol);
                                 else
                                   strcpy(szWork, &szCol[2]);
                               }
                           }
                       s2sloadRec->putField(columnListField + nProductCounter, szWork);
                     }

                   nProductCounter++;
                  } // if was last button of a product

                 // If button is being vacated, update vacated buttons counter.
                 if (button[x].removed)
                   nVacatedButtonCounter++;

                } // for each button

               s2sloadRec->putField("Old days left", oldMinDays);
               ASSERT(s2sloadRec->lastError == PXSUCCESS);
               s2sloadRec->putField("New days left", newMinDays);
               ASSERT(s2sloadRec->lastError == PXSUCCESS);

               nTmp = 1; // init space to sales flag to true
               machloadRec->putField("S2S", nTmp);
               ASSERT(PXSUCCESS == machloadRec->lastError);

               // Update machload rec.
               curMachLoad.updateRec(machloadRec);
               // Append Sp2SLoad rec to table.
               curSp2SLoad.appendRec(s2sloadRec);
              }

            } // if min days increased enough
          } // if curFacility.lastError == PXSUCCESS
        } // if curMachStat.lastError == PXSUCCESS
```

```
                curVenderLoad.gotoNext();

             } // while curVenderLoad.lastError == PXSUCCESS

          // write out total additions for sp 2 sales
          for (x = 0; x < nProductCount; x++)
             {
                if (stProductName[x].nAdds > 0)
                   {
                     totalRec->putField("Route", routeIdent);
                     totalRec->putField("Product code", stProductName[x].nCode);
                     totalRec->putField("Product name", stProductName[x].szName);
                     totalRec->putField("Count", stProductName[x].nAdds);
                     curSp2STotal.appendRec(totalRec);
                   }
             } // for
        } // if all cursors created successfully

      curFacility.close();
      curMachStat.close();
      curVenderLoad.close();
      curMachLoad.close();
      curSp2SLoad.close();
      curSp2STotal.close();


      CHECKHANDLES();
      /////////////////////////////////////////////////////////////
    } // GenerateS2SA()
```